Intelligent intrusion detection in low power IoTs

Saeed, Ahmed; Ahmadinia, Ali; Javed, Abbas; Larijani, Hadi

Published in:
ACM Transactions on Internet Technology

DOI:
10.1145/2990499

Publication date:
2016

Document Version
Author accepted manuscript

Intelligent Intrusion Detection in Low Power IoTs

Ahmed Saeed, School of Engineering and Built Environment, Glasgow Caledonian University, UK
Ali Ahmadinia*, Department of Computer Science, California State University San Marcos, US
Abbas Javed, Hadi Larijani, School of Engineering and Built Environment, Glasgow Caledonian University, UK

Security and privacy of data are one of the prime concerns in today's internet of things (IoT). Conventional security techniques like signature-based detection of malware and regular update of signature database are not feasible solutions as they cannot secure such systems, having limited resources, effectively. Programming languages permitting immediate memory accesses through pointers often result in applications having memory-related errors, which may lead to unpredictable failures and security vulnerabilities. Furthermore, energy efficient IoT devices running on batteries cannot afford the implementation of cryptography algorithms as such techniques have significant impact on the system power consumption. Therefore, in order to operate IoT in a secure manner, the system must be able to detect and prevent any kind of intrusions before the network (i.e., sensor nodes and base station) is destabilized by the attackers. In this paper, we have presented an intrusion detection and prevention mechanism by implementing an intelligent security architecture using Random Neural Networks (RNN). The application's source code is also instrumented at compile time in order to detect out-of-bound memory accesses. It is based on creating tags, to be coupled with each memory allocation and then placing additional tag checking instructions for each access made to the memory. To validate the feasibility of the proposed security solution, it is implemented for an existing IoT system and its functionality is practically demonstrated by successfully detecting the presence of any suspicious sensor node within the system operating range and anomalous activity in the base station with an accuracy of 97.23%. Overall, the proposed security solution has presented a minimal performance overhead.

CCS Concepts: •**Security and privacy** → **Intrusion/anomaly detection and malware mitigation;** •**Networks** → *Sensor networks;*

Additional Key Words and Phrases: IoT security, data integrity, code instrumentation, illegal memory accesses, buffer overflows, neural networks

ACM Reference Format:
Ahmed Saeed, Ali Ahmadina, Abbas Javed and Hadi Larijani, 2016.Random Neural Network based Intelligent Intrusion Detection and Prevention Mechanism for IoT Applications *ACM Trans. Internet Technol.* 0, 0, Article 0 (2016), 26 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The tremendous growth of the IoT devices has changed the world of embedded systems and the availability of low-cost typical IoT components has enabled designers to connect "things", including but not limited to industrial sensors, smart phones, medical devices, household appliances, cars and other vehicles to the internet. There are two important key characteristics that make such systems prone to the security attacks. Firstly, the simplified processing capabilities and limited power resources expose them to a number of possible security attacks. Secondly, the network connectivity to the outside world, without any inbuilt protection, also leaves such systems vulnerable to

*Corresponding Author: Ali Ahmadinia, Email: aahmadinia@csusm.edu

security attacks. In a typical IoT system, these vulnerabilities can be exploited by an attacker to steal private data, drain the power supply, destroy the system, or modify the system behaviour for other than its designed purpose as shown with an example of smart home in Fig. 1. Furthermore, advancements in communication technology has also resulted in direct machine-to-machine (M2M) communication for example in smart electric meters, remote controlled devices and wireless sensor nodes. As system components are interconnected and also accessible via internet, they are at increased risk of security attacks. Therefore, security-awareness is becoming a primary design objective to be considered at each level of the software and hardware platforms design for future IoT devices. The full advantages of the IoT cannot be realized without ensuring the security of all the connected devices [Miorandi et al. 2012; Xu et al. 2014].

Different system requirements such as high throughput, low power consumption and higher level of security must be considered during the design phase. For example, IoT devices with direct power supply connection do not consider energy efficiency as a prime concern. However, most of the IoT nodes are micro-controller based embedded devices relying on batteries where reducing energy consumption is a top priority. Normally such devices have zero security due to limited resources and therefore cannot afford security solutions like anti-virus and cryptography [Trappe et al. 2015]. In such IoT devices, the information transmitted by the nodes can be interfered and the application running on the micro-controller can also be compromised. In this way, at least the application data will be corrupted which results in its incorrect execution. We will demonstrated in our case study that without proper security mechanism, the wireless sensor nodes can be manipulated and the base station can be compromised easily if the attacker is familiar with the communication protocol and hardware architecture.

In the traditional IoT devices, the confidentiality and integrity of data are achieved by end-to-end encryption as the M2M communication cannot be accessed by other than authorized nodes as it will be always encrypted. For battery operated IoT devices, such encryption based security features are not a feasible solution as the encrypted communication has a significant impact on the system power consumption. In such power constrained systems, without encryption support, the software must be able to detect and report any malicious activity within the system.

The vulnerabilities in the application, running on a IoT device, can also be exploited and manipulated by software-based attacks such as through buffer overflows. For instance, programming languages like `C` and `C++` are commonly used by the embedded system software designers as they provide a powerful set of features such as low memory footprint, little run-time support, low-level direct memory accesses and arithmetic operations through pointers. At the same time, illegal memory accesses (IMAs) such as out-of-bound buffer accesses are also major concerns in applications written with such programming languages. Normally, the starting address is assigned to a pointer when a memory area of required size is allocated, whereas an access is considered legal only when either its actual pointer or a pointer derived from it is used between the allocation and deallocation of a specific memory area. For example, memory accesses can happen outside the intended range if the index calculation of an array is based on an erroneous formula. It is almost impossible to detect and diagnose such behaviour using static analysis-based tools. Even when an application is tested intensively through these tools, such bugs can still exist as it is practically impossible to create all of the input combinations for an error to occur in the development or test phase.

Without required protection, many security threats like viruses, trojans and worms can modify application data by gaining illegitimate access to secured blocks of the memory. Software-based attacks have become increasingly widespread and buffer overflow is one of the major causes of such security outbreaks. According to a report published by Sourcefire [Younan 2014], buffer overflow based attacks are responsible

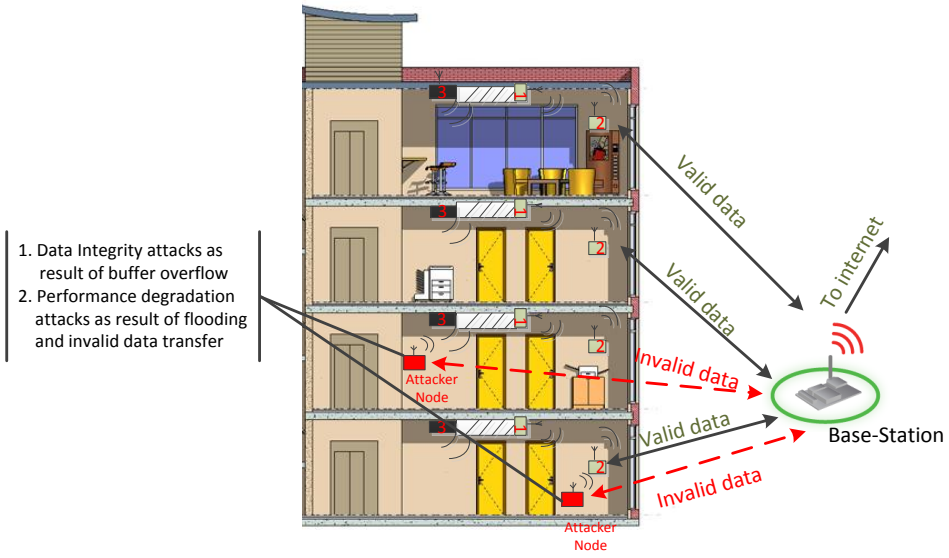

Fig. 1: Attack scenarios in a typical wireless sensor nodes based smart home IoT System

for 14% of all and 35% of critical vulnerabilities over the past 25 years. Stack smashing [One 1996] is a classic example of buffer overflow based attacks. During attack execution, the return address of a function on the stack is replaced through buffer overflow. In case of unprotected execution, on function return, the control may be switched to the specific location where malicious code is placed as shown in Fig. 2. In order to conduct this kind of attack, an unprotected buffer variable is located in the program and then it is loaded with a special input value so that its stack frame is overflown and the return address changed to jump to a new location. Likewise, the buffers allocated either dynamically in the *heap region* or globally in the *data region* of the memory can be also overflown and exploited by the attackers.

Similarly, performance degradation attacks such as denial of service (DOS), flooding, jamming and battery drainage attacks bring down the system performance by overloading the computing and communications resources. The unnecessary resource utilization downgrades the operability of the system and may implicate real-time behaviours of the system. IoT devices operating on battery are target of these types of attacks. Such attacks may also involve continuous sending of requests to the victim base station, in order to force execution of power-hungry tasks. Such type of attacks also targets communication medium of the system in order to reduce throughput which may result in failure to meet specific deadlines especially in the case of real-time systems. The system can only be considered secured if it has the ability to detect any abnormality and to stop its propagation preventing further damage to the system operability.

In this paper, a fast and effective two-layer anomaly based intrusion detection and prevention mechanism (IDPM) is presented to detect and prevent a wide range of data integrity and performance degradation attacks in the IoT systems. The fist layer of the IDPM is based on learning normal behaviour of the system using RNN taking diverse dataset, covering both valid and invalid cases, as input parameters. The trained RNN model is then embedded in the base station of the IoT system to detect any anomalous behaviour and prevent its propagation. The second layer of the IDPM is designed to

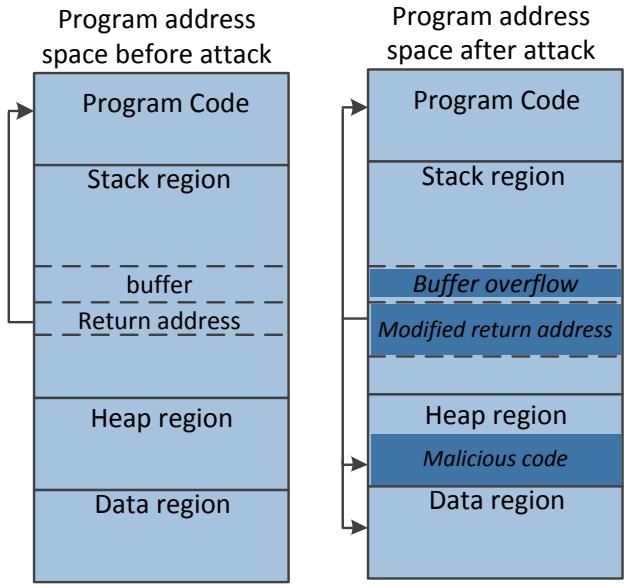

Fig. 2: Typical buffer overflow leading to stack smashing or ROP attack

detect a wide range of IMA bugs and data integrity attacks dynamically including out-of-bound read and write accesses, stack-overflows, stack-underflows, heap-overflows, heap-underflows, overflows and underflows in globally defined variables (data and bss segments) and direct-indexed overflows/underflows. The proposed solution also acts as a health monitoring system for the IoT sensor nodes by analysing data being transmitted to the base station. As in case of any malfunction, the valid sensor node may either stop its operation and or transmit invalid data to the base station. The RNN model has been trained to detect such cases as an intrusion and report them to the main server. The proposed solution effectiveness and performance overhead is measured for an existing IoT system consisting of sensor nodes transmitting data to a base station. Through experimental setup it is shown that without proper security mechanism it is possible to intrude into the application running on the base station. Furthermore, it is also demonstrated that the base station successfully detected the presence of the malicious sensor node when the given IoT device is enabled with the proposed IDPM.

The paper is organized as follows: The current work related to the existing intrusion detection techniques is summarized in Section 2. The general approach and implementation details of the proposed intrusion detection and prevention mechanism for an existing IoT system are described in Section 3. Section 4 presents the effectiveness and performance evaluation while Section 5 concludes this paper.

2. RELATED WORK

Intrusion detection has been studied widely in the perspective of computer networks as they have been the prime target of security attacks. It is also becoming one of the key research areas in the field of embedded and IoT systems due to their increasing functionality and connectivity. Different solutions have been proposed to detect and prevent security attacks either through dedicated hardware modules or software-based mechanisms. Currently, various security techniques have been presented in the literature such as reference monitors, cryptography, dynamic information flow track-

ing (DIFT), neural network based intrusion detection solutions and compile-time code instrumentation.

Most of the reference monitor related solutions are based on scanning processor-executed code by comparing it to a predefined model and the security subsystem operates in parallel with each processor such as presented by Kornaros and Pnevmatikatos [Kornaros and Pnevmatikatos 2013]. The hardware-assisted security monitors [Mao and Wolf 2010; Rahmatian et al. 2012; Yoon et al. 2013] are based on the concept of sensing deviation in program execution at run-time by comparing behaviour against a static model for the purpose of detecting code modification attacks. Mao and Wolf [Mao and Wolf 2010] have used hash-based patterns for basic blocks comprising of fewer instructions. Similarly, Yoon et al. [Yoon et al. 2013] presented a security solution for multicore processor based real-time embedded systems that is based on running monitoring program on a dedicated core.

Security attacks that target confidentiality and integrity of the data can be avoided by encrypting the data. Various security technologies and mechanisms have been designed around cryptography algorithms in order to provide specific security services. Cryptography algorithms such as Advanced Encryption Standard (AES) [NIST 2001] and RSA [RSA 2003] have been used in embedded systems to establish integrity of data within the system. In a recent survey [Granjal et al. 2015], various security mechanisms for the IoT devices have been discussed based on cryptography. However, such techniques are not realistic for low-power IoT devices due to resource constraints and large power consumption overheads.

Dynamic Information Flow Tracking (DIFT) [Doudalis et al. 2012; Suh et al. 2004; Schwartz et al. 2010] is another effective technique to ensure protection against software-based attacks. The basic idea is to mark certain input sources that are untrusted and track the flow of information that is being supplied to the program through these input sources. All data values that are dependent on the marked input value are also get marked (more commonly known as tainted values). If the tainted value leads to any un-tainted data value during program execution then alert signal is generated reflecting an unauthorized access.

The hardware-based security solutions [Mao and Wolf 2010; Rahmatian et al. 2012; Yoon et al. 2013], are not generic and require either dedicated hardware modules or specific modifications in the processor pipeline or cache architecture. Similarly, DIFT based hardware-assisted solutions such as Secure Program Execution [Suh et al. 2004], Dynamic Tainting [Doudalis et al. 2012; Suh et al. 2004; Schwartz et al. 2010] are based on tagging data coming from untrusted sources and then tracking their usage as the application executes. These techniques require modifications in application's data, processor architecture and memory layout. Therefore, they are not feasible for low-cost IoT systems.

Intrusion detection techniques [Amin et al. 2009; Raza et al. 2013] for IP based sensor networks would require implementation of TCP/IP stack and signature storage which is not possible for power and resource constrained microcontroller based sensor nodes. Similarly vector based classification methods [Li et al. 2014] require storing of valid dataset on the system and then performing intrusion detection analysis which is again not feasible due to higher memory footprint and performance overhead.

Different anomaly based intrusion detection techniques have been proposed in the literature based on statistical modelling of system behaviour. Machine-learning techniques have been utilized by learning a model, depicting both normal and anomalous behaviour of the system. Moreover, application data can also be protected by inserting run-time checks in the application code in order to detect illegal memory accesses (IMAs). Existing anomaly detection solutions based on neural networks and compiler-

based techniques for the detection of IMAs have been presented in the following subsections.

2.1. Neural Networks based Intrusion Detection

Neural networks have been deployed in different scenarios, covering pattern classification, function approximation, prediction, optimization and control theory. Neural networks have been used to detect anomalies in the program behaviour by classifying system-call sequences [Han and Cho 2005]. Recent surveys [Bhuyan et al. 2014; Butun et al. 2014] have presented an extensive overview of anomaly detection techniques and showed that the neural networks have been tested and implemented in the field of network systems largely. Wu and Banzhaf. [Wu and Banzhaf 2010] has presented a comprehensive overview of intrusion detection techniques using computational intelligence methods such as fuzzy systems, evolutionary neural networks, artificial immune systems, artificial neural networks (ANN) and swarm intelligence. Callegari et al. [Callegari et al. 2014] have used neural networks to predict the flow of traffic in a given time window within a network and detect any intrusion in the system. They have used a multilayer feed-forward architecture for this purpose.

Viera et al. [Vieira et al. 2010] have used ANN in the cloud environment for anomaly based intrusion detection. They have used a large feature vector for training the model and presented that the ANN can be used to detect intrusions more effectively. The detection accuracy of ANN based models is dependent on the input feature vector and number of hidden layers used in the training phase. Furthermore, the detection phase of such intrusion detection systems is computationally intensive and requires more time to detect any anomaly. Gelenbe [Gelenbe 1989; 1990] proposed the new class of ANN as RNN which is based on concepts of probability theory applied to Markovian queuing theory. In the literature, RNN has been referred to as a G-network [Gelenbe 1991]. RNN is easy to implement on hardware as its neurons can be represented by simple counters [Abdelbaki et al. 2000; Mohamed and Rubino 2002]. Mohamed and Rubino. [Mohamed and Rubino 2002] compared RNN with ANN and showed that training time for RNN is greater than ANN, but RNN outperformed ANN during run-time phase in total calculation time. They further showed that RNNs have strong generalisation capability for the patterns not covered in the training phase. Applications of RNN have been reported for modelling, optimisation, pattern recognition, classification, and communication [Timotheou 2010].

Different neural network architectures have been proposed depending on the application under consideration but the overall efficiency is largely dependent on the learning algorithm being used in the training phase [Alarcon-Aquino et al. 2006]. For example, the gradient descent (GD) algorithm was introduced by Gelenbe [Gelenbe 1993] for recurrent RNN, which can be applied to feed forward RNN model. Likas and Stafylopatis [Likas and Stafylopatis 2000] proposed another learning algorithm based on minimisation of quadratic error function using quasi-Newton optimisation technique. The learning algorithm for multiple class RNN is introduced by extending the GD algorithm for single class of RNN which is applicable on feed forward and recurrent RNN [Gelenbe and Hussain 2002]. The complexity of learning algorithm is $O(nC^3)$ for recurrent RNN and $O(nC^2)$ for feed forward RNN where n is the number of neurons and C is the number of signal classes. Timotheou [Timotheou 2008] proposed the non-negative least square (NNLS) learning algorithm for RNN by approximating the RNN equations and showed that his algorithm is better than the GD algorithm.

The majority of researchers have used the GD algorithm for learning the weights of the RNN model. The GD algorithm is easier to implement, but zigzag behaviour may occur near the local minimum and in case of problems with multiple local minima. The evolutionary algorithms are more suitable for solving the optimisation problem.

These techniques are better than gradient-based techniques as they do not require the calculation of derivatives and they do not get stuck in local minimum (the major problem with GD algorithm).

Georgiopoulos et al. [Georgiopoulos et al. 2011] have used the differential evolution (DE) and particle swarm optimisation (PSO) algorithms for training of RNN and compared with the GD algorithm. Aguilar and Colmenares [Aguilar and Colmenares 1998] have implemented the hybrid training algorithm for RNN by integrating genetic a algorithm with GD algorithm. They trained the RNN with GD algorithm and optimised the weights with genetic algorithm. The results showed that hybrid algorithm is better than GD algorithm.

The existing neural networks based intrusion detection solutions [Vieira et al. 2010; Mohamed and Rubino 2002] are computationally intensive, rely on extensive profiling of the communication traffic and have been designed for such systems having ample resources where power consumption is not a design constraint. Therefore, such solution cannot be deployed for battery operated wireless sensor nodes based IoT systems. On the other hand, the first security layer of our proposed solution is based on a RNN model which is easy to implement, has low memory footprint, consumes minimal power and can detect any deviation in the system behaviour effectively.

2.2. Compile-time Code Instrumentation

Different solutions have been proposed to detect memory errors in C/C++ based applications. As mentioned earlier, static analysis tools do not provide 100% guarantee, and hence they have been substituted with dynamic techniques. Many software-based dynamic memory bug detection techniques have been proposed in the literature that vary in implementation level, memory utilization, run-time overhead, types of bugs detected, probability to detect bugs, supported architectures and many other features. In this section, we have discussed only those solutions that are similar to our proposed compile-time instrumentation solution.

Referent-object based approaches [Jones and Kelly 1997; Ruwase and Lam 2004; Younan et al. 2010; Akritidis et al. 2009] work at source-code level by maintaining a separate table, using different data structures, to record bounds of each memory allocation. This table is then used to verify memory accesses by performing table lookups at run-time. These techniques differ in the implementation and handling of record tables. Avijit et al. [Avijit et al. 2004; Avijit and Gupta 2006] have implemented LibsafePlus and TIED. Static allocations are handled by TIED whereas LibsafePlus deals with dynamic information about the stack size and heap allocations. These details are then used at run-time to detect any overflow. Furthermore, these solutions require customised memory allocator libraries to achieve reduced performance overhead. SAFECode [Dhurjati et al. 2006] which is also an object-based approach, operates at source-code level. It instruments loads and stores to prevent illegal memory accesses by using points-to analysis and type-inference to find type-safe regions of the heap. Similarly, PAriCheck [Younan et al. 2010] computes bounds and assigns label to each fixed-size memory block and stores these labels in a separate table at run-time. On each pointer arithmetic operation, the label is compared by values in the table.

On the other hand, the pointer-based approaches [Necula et al. 2005; Nagarakatte et al. 2009; Hasabnis et al. 2012] associate base and size metadata with every pointer and insert run-time checks manipulating metadata information during load/store operations of pointer values. CCured [Necula et al. 2005] combines instrumentation with static analysis to insert run-time checks and removes redundant checks at compile time. CCured fails to work when un-instrumented pre-compiled libraries are in use. Unlike prior pointer-based (also known as fat pointer) approaches that modify pointer representations and object layouts [Hasabnis et al. 2012], SoftBound [Nagarakatte

et al. 2009] records the bounds information in a disjoint metadata which is accessed via explicit table lookups on loads and stores of pointer values only. AddressSanitizer [Serebryany et al. 2012] verifies whether each allocated memory block is safe to access by creating shadow memory around stack and global objects to detect overflows. The shadow memory is checked on each load and store request.

Each of the above mentioned memory bug detection tools has its own strengths and limitations. For example, tools that operate at binary level detect bugs effectively without any source code requirement but at the cost of more execution time overhead. As discussed earlier, the tools that operate at source-code level also require either modified run-time memory allocators or a dedicated compiler driver as it is the case in several existing solutions [Doudalis et al. 2012; Dhurjati et al. 2006; Serebryany et al. 2012; Akritidis et al. 2009; Younan et al. 2010]. Other software based techniques such as LBC [Hasabnis et al. 2012] have presented much lower performance overhead but they require modifications in the source code, thus presenting compatibility issues. While our solution also requires source code for instrumentation, it does not need customized run-time libraries and static analysis. Our technique presents higher detection rate by instrumenting all buffer allocations including buffers allocated inside `struct` type data variables which are left undetected by the existing solutions excluding SoftBound.

Unlike SoftBound, the second security layer of our proposed solution is a combination of object and pointer-based approach. We create tag marks for each memory object and propagate these tag marks to all the pointers that are associated with that memory object. Moreover, we do not perform any table-lookup search at run-time which makes our solution more efficient. Existing solutions are able to detect temporal safety errors but with much higher performance overhead. On the contrary, lower performance and power consumption overhead has been reported by our solution when tested in a IoT system.

3. SYSTEM ARCHITECTURE

Detecting abnormality based on behaviour analysis involves learning of the normal operation of the system and identification of any event that deviates from the previously learned model. In this way, unknown security attacks can also be detected which normally left undetected by the signature-based techniques. The proposed intrusion detection and prevention mechanism (IDPM) is enabled with two layers of security. In the first layer, the Random Neural Network (RNN) is used to detect any abnormality in the behaviour of the system. In the second layer, a compile-time code instrumentation approach is used to detect illegal memory accesses (IMAs) at run-time.

A previously designed smart controller [Javed et al. 2015] IoT system, as shown in Fig. 3, has been used to implement the proposed IDPM. The system comprises of a base station, sensor nodes and a web server. This has been designed using multiple RNN models, which have been implemented on low cost Arduino boards. The base station is developed on an Arduino Mega board and it is connected with control panel of the environment chamber to turn on/off heater, cooler, and ventilation. Each sensor node is connected with RFM 69 W transceiver to transmit the data to base station in the form of a string with the following format [node ID, CO2, temperature, humidity, light, and intensity (optional)]. This smart controller is capable of detecting and estimating the number of occupants inside the building in order to turn on/off the Heating, Ventilation and Air Conditioning (HVAC) control. In addition, it can interact with occupants to maintain the occupant preferred set points for heating and cooling. The application running on the base station processes data after receiving it from the sensor nodes through a transceiver and further communicates with a web server through a on-board

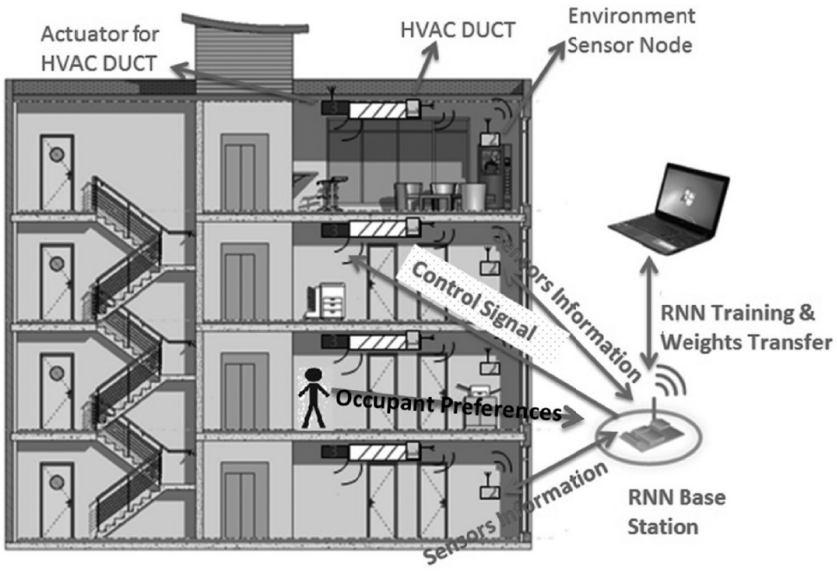

Fig. 3: Smart Controller IoT System [Javed et al. 2015]

WiFi module. The sensor nodes are battery operated and cannot afford to send data in encrypted form as it consume considerable amount of power.

Although the proposed IDPM solution has been implemented and tested for a centralized system (consisting of a base station and wireless sensors) but this can be adopted to other scenarios as well. For distributed systems, the same approach can be incorporated by implementing the proposed IDPM inside each IoT device that receives data but in such cases this will require separate training phase for each of the devices.

3.1. Layer-1: RNN based Intelligent anomaly detection

3.1.1. Random Neural Networks. The proposed solution has been designed keeping in mind the scalability of the specific IoT system. For instance, when large number of wireless sensor nodes (i.e. more than 10) are communicating with the base station it is not feasible to implement simple solutions (such as if-else conditional statements are computationally intensive). Moreover, during experimental testing of the given IoT system it is learned that threshold checks cannot applied against each sensor node at the base-station. Thus, it has become necessary to generate a predictive model considering all the input parameters being received by the base station from each node.

First we need to understand how RNN works in oder to adjust it for anomaly detection. In the RNN, as shown in Fig. 4, signal travels in the form of impulse between the neurons. If the receiving signal has a positive potential (+1) it represents excitation, and if the potential of the input signal is negative (-1) it represents inhibition to the receiving neuron. Each neuron, i, in the RNN has a state $k_i(t)$ which represents the potential at time t. This potential $k_i(t)$ is represented by non-negative integer. If $k_i(t) > 0$ then neuron, i, is in excited state and if $k_i(t) = 0$ then neuron i is in idle state. When neuron, i, is in excited state, it transmits impulse according to the Poisson process rate r_i. The transmitted signal can reach neuron, j, as excitation signal with probability $p^+(i, j)$ or as inhibitory signal with probability $p^-(i, j)$, or can leave the network with

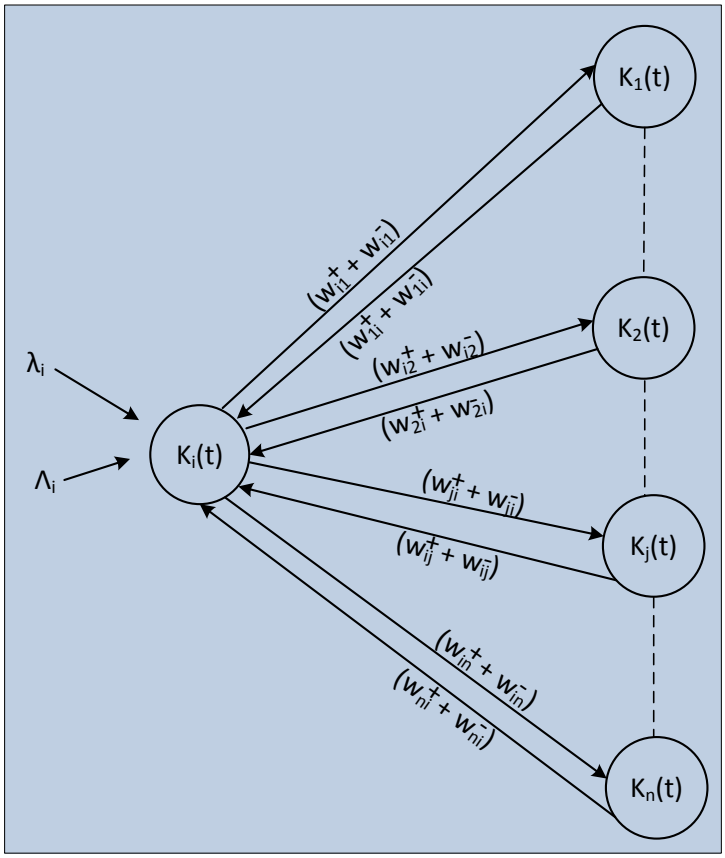

Fig. 4: Random Neural Network

probability $d(i)$ such that

$$d(i) + \sum_{j=1}^{N} \left[p^+(i,j) + p^-(i,j)\right] = 1 \forall i \tag{1}$$

$$w^+(i,j) = r_i p^+(i,j) \geqslant 0 \tag{2}$$

$$w^-(i,j) = r_i p^-(i,j) \geqslant 0 \tag{3}$$

combining Equation 1- 3

$$r(i) = (1 - d(i))^{-1} \sum_{j=1}^{N} \left[w^+(i,j) + w^-(i,j)\right] \tag{4}$$

The firing rate between the neuron is represented by $r(i) = \sum_{j=1}^{N} [w^+(i,j) + w^-(i,j)]$. As *'w'* matrices are the product of firing rate and probabilities, therefore these matrices always hold non-negative values. External positive or negative signal can also reach neuron i at poisson rate Λ_i and λ_i respectively. When positive signal is received at neuron i its potential $k_i(t)$ will increase to +1. If neuron i is in excitation state and it

Table I: Description of RNN symbols

RNN Symbols	Description
q_i	Probability neuron i excited at time t
$p^+(i,j)$	Probability neuron j receives positive signal from neuron i
$p^-(i,j)$	Probability neuron j receives negative signal from neuron i
r_i	Firing rate of neuron i
Λ_i	Arrival rate of external positive signals
λ_i	Arrival rate of external negative signals
$d(i)$	Probability a signal from neuron departs from the network
$k_i(t)$	Potential of neuron i at time t

receives negative signal the potential of neuron i will decrease to zero. Arrival of negative signal will have no effect on neuron i if its potential is already 0. The description of symbols used is given in Table I.
Consider the vector $\mathbf{K}(\mathbf{t})= (k_1(t), ...k_n(t))$ where $k_i(t)$ is the potential of neuron i, n is the total number of neurons in the network and $\mathbf{K}$ is continuous time Markov process. The stationary distribution of $\mathbf{K}$ is represented as:

$$\lim_{t\to\infty} Pr(K(t))) = (k_1(t).....k_n(t)) = \prod_{i=1}^{n}(1-q_i)q_i^{ni} \tag{5}$$

For each node i

$$q_i = \frac{G_i^+}{r_i + G_i^-} \tag{6}$$

where

$$G_i^+ = \Lambda_i + \sum_{j=1}^{N} q_j w^+(j,i) \tag{7}$$

$$G_i^- = \Lambda_i - \sum_{j=1}^{N} q_j w^-(j,i) \tag{8}$$

For three layer network, q_i for each layer is calculated as

$$q_{i\varepsilon I} = \frac{\Lambda_i}{r_i + \lambda_i} \quad where\ I is\ input\ layer \tag{9}$$

$$q_{h\varepsilon H} = \frac{\sum_{i\varepsilon I} q_i w^+(i,h)}{r_h + \sum_{i\varepsilon I} q_i w^-(i,h)} \quad where\ His\ hidden\ layer \tag{10}$$

$$q_{o\varepsilon O} = \frac{\sum_{i\varepsilon H} q_h w^+(h,o)}{r_h + \sum_{i\varepsilon I} q_h w^-(h,o)} \quad where\ Ois\ Output\ layer \tag{11}$$

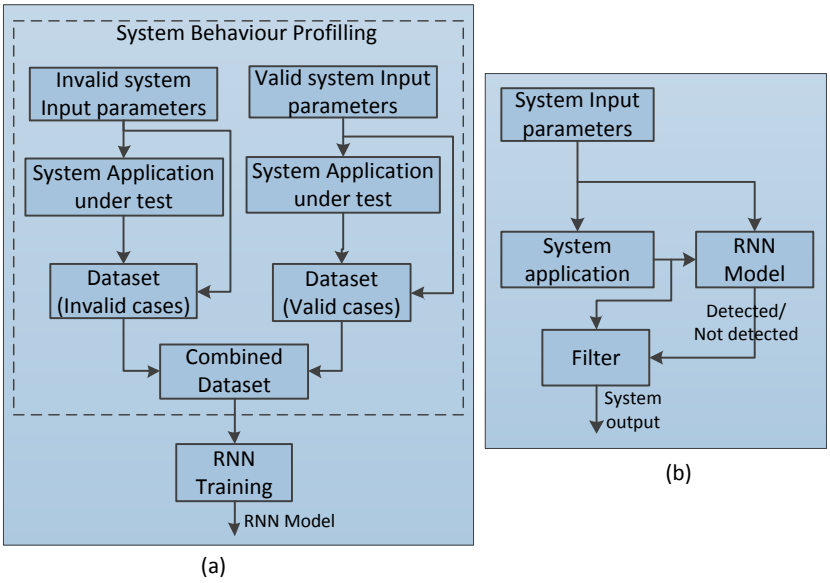

Fig. 5: Layer-1 design and implementation of the proposed IDPM: (a) Feature dataset acquisition and training of RNN model. (b) Trained RNN model implemented within the system.

3.1.2. Training data for RNN. The intelligent intrusion detection solutions are based on generating prediction model using the extracted feature dataset. In our case, the inputs and outputs of the system under test are profiled in the form of a feature dataset presenting behaviour of the system under normal conditions and invalid inputs respectively. The RNN is then used to train a predication model classifying the obtained feature dataset as either anomalous or normal. Invalid input parameters provided to the system under test and output generated as a result are considered as a sign of potential anomaly in the system behaviour. The design and implementation of the layer-1 of the proposed IDPM is presented in Fig. 5. To get the feature dataset, the application running on the main system can be profiled as shown in Fig. 5(a). The feature dataset is then used to train the RNN model as shown in Fig. 5(b).

To verify the functionality of the proposed intrusion detection mechanism, the feature dataset is obtained for the given smart controller IoT system. The valid feature dataset is constructed by profiling the smart controller application, representing the system behaviour under normal conditions. For this purpose, outputs generated by base station transceiver in the form of node id, temperature, CO_2 and humidity values are stored in the dataset. Furthermore, the execution time of smart controller application running on the base station for the single event as well as aggregate execution time for a specific period of time are stored in the dataset.

3.1.3. RNN Predictive Model Design. During the training phase, valid ranges of the outputs generated by the base station transceiver are recorded which have been used to determine the invalid cases. It is important to note here that we have added only few invalid test inputs for training purpose in order to distinguish between normal and abnormal system behaviour for the RNN model. For testing purposes, the performance of RNN predictive model is evaluated for the patterns not included in the training dataset.

After getting the comprehensive feature dataset, the next step is learning of system behaviour in the form of RNN prediction model. Here, the main goal of the training is to learn the input and output relationship by adjusting the interconnections weights of RNN. In our previous work [Javed et al. 2015], we have presented the benefits of using a hybrid particle swarm optimisation with sequential quadratic programming (PSO-SQP) algorithm for training the RNN. For instance, the PSO converges very quickly to global minima but it gets very slow on local minima whereas, SQP optimisation algorithm can be used for fine tuning if feasible starting points are provided. In this work, we have used the same algorithm and trained a RNN prediction model, as shown in Fig. 6, having six nodes in the input layer, six nodes in the hidden layer and one node in the output layer. We trained the RNN model using different numbers of hidden neurons keeping in mind its implementation cost. For instance, when using a lower number of hidden neurons (i.e. 3, 4, 5 neurons), the trained model accuracy deteriorated whereas when a higher number of hidden neurons (i.e. 7, 8, 9 neurons) is used, the detection accuracy remained almost the same but at the expense of increase in its implementation cost. In our case, the PSO algorithm slowed down the training phase around 100 iterations when the obtained feature dataset is used. For this reason, the PSO algorithm has been used for first 100 iterations to get initial weights values and then SQP optimization algorithm has been further used for fine tuning. The mean squared error (MSE) achieved by PSO is 0.134 while MSE achieved by PSO-SQP is 0.104. Achieving lower MSE has resulted in higher accuracy to detect intrusions.

Although, using more than one output neuron improves the training time of RNN model but it increases the implementation cost resulting overhead in application code size and power consumption. When using two output neurons, the RNN model assigned same number of weights to each output which doubled the implementation cost. Furthermore, in our previous work [Javed et al. 2015; Javed et al. 2016], we have successfully demonstrated that a single output neuron can be used to represent the given dataset. The over-fitting problem is avoided by defining the valid limits of each input parameter given to the RNN model. For instance, the particular IoT system under test, we have gathered dataset covering valid variations in supervised mode (changes in Co2, humidity and temperature with respect to weather) and any sudden change in these features will correspond to the intrusion in the system.

The base station architecture enabled with layer-1 of the proposed IDPM is shown in Fig. 7. The RNN model output predicts whether the system is working normally or it has been intruded by generating the `enable` signal. In case of any intrusion, the `enable` signal will alert the *filter* module to stop receiving data from sensor nodes. The *filter* module will disable the smart controller application and send the pre-defined data as output for the HVAC. Furthermore, an alert signal will be sent to the web server indicating occurrence of anomalous activity at the base station. In this way, the intrusion will be prevented and system will operate in default mode.

The main drawback of anomaly based intrusion detection techniques is they might generate false alarms if the normal behaviour of the system changes over a period of time. In such cases, it is necessary to update the normal profile periodically. For systems, as in our case, that do not change the normal behaviour rapidly, the anomalous activity can be detected effectively. The learned model must be updated periodically when the normal system behaviour changes. In this way the intrusion detection accuracy can be improved.

The design and implementation of the second security layer is presented in next subsection whereas the effectiveness and overheads of the proposed solution are discussed in the Section 4.

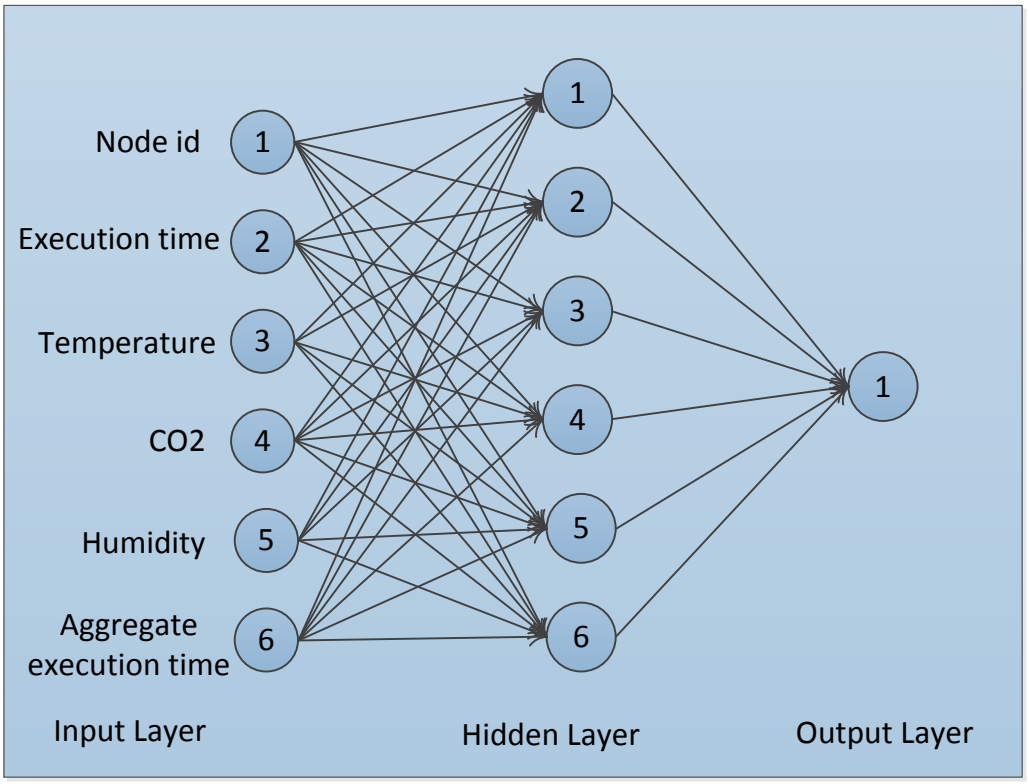

Fig. 6: Architecture of the RNN model

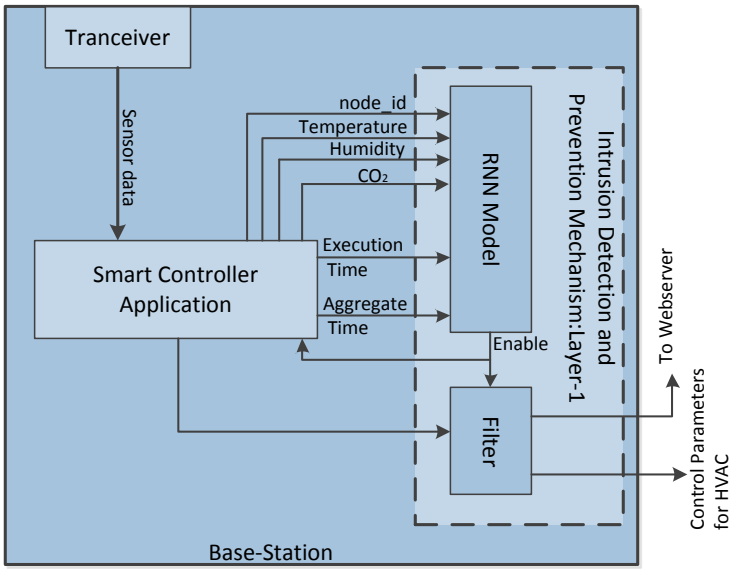

Fig. 7: IoT System protected with layer-1 of the proposed intrusion detection and prevention mechanism (IDPM).

3.2. Layer-2: Compile-time Code Instrumentation

This security layer instrument the source code of the application running on the base station. It is based on storing the memory objects bounds information in separately allocated tag marks and then inserting tag checking instructions. For example, when a memory object created and a certain memory area is reserved for it, our technique creates two tag marks *tag_start* and *tag_end*. The base address and the end address of that memory object are calculated and stored in the *tag_start* mark and the*tag_end* mark respectively. Finally, when that memory object is accessed, the run-time check instructions compare the address being accessed with the bounds stored in its associated tag marks.

Our solution is compatible with C/C++ source code as it does not change the memory layout and tag marks are generated and maintained separately. The source code of the application is converted into an intermediate representation form which is used to detect each memory allocation, create tag marks and insert run-time checks. The typical memory layout of a C/C++ program is shown in Fig. 8(a). For each memory object, whether it is globally, statically or dynamically allocated, the tag marks are created and memory object bounds are assigned to these tag marks as shown in Fig. 8(b). The record of tag marks along with their associated memory objects is kept separately in a specialized record table as depicted in Fig. 8(c). The tag marks are also created for sub-objects that are defined inside a structure memory object. In that case, our solution stores the sub-object information along with the main memory object in third column of the record table. It should be noted that the record table is used at the time of code instrumentation only to place relevant check instructions and it is not part of the final executable file. Unlike SoftBound, which uses record tables for bounds lookup at runtime, our solution utilizes this record table at compile time only and are deleted before generating final instrumented executable. Using this record table, the tag checking instructions will be inserted before each load or store instruction detected for the corresponding memory object. During tag address comparison, the given memory access will be considered legal only if the address used to access the memory area is greater than the address stored in its *tag_end* mark and less than the address contained by its *tag_start* mark. In case of any spatial IMA bug, the addresses stored in tag marks will be surpassed and tag check instructions will raise bug alarm and abort the application. The design of layer-2 of the proposed IDPM is based on the following steps.

Step-1. Function Duplication Enabling Inter-procedural Tag propagation: When memory objects are accessed by the pointers and these pointers are passed as function arguments then the corresponding tag marks must also be propagated with them as well. One solution is to allocate all the tag marks globally so that the tag marks can be accessed anywhere in the program without modifying the function arguments. However during testing, this approach failed when functions are called recursively. To handle this problem, the tag marks are created in the same memory segment where corresponding memory object is being allocated. Moreover, the function duplication technique is used to create a copy of the function and additional arguments are added to pass tag_start and tag_end marks for each pointer argument. In order to mark the function being duplicated the actual function name is used along with a unique attribute. The details of each function being duplicated is saved in a separate table that is used in later stages to replace function call instructions.

Step-2. Tag Creation for Global Memory Objects: For globally defined memory objects that are created statically, such as buffers and structures, the memory is allocated directly at program entry level in *data* and *bss* segments as shown in

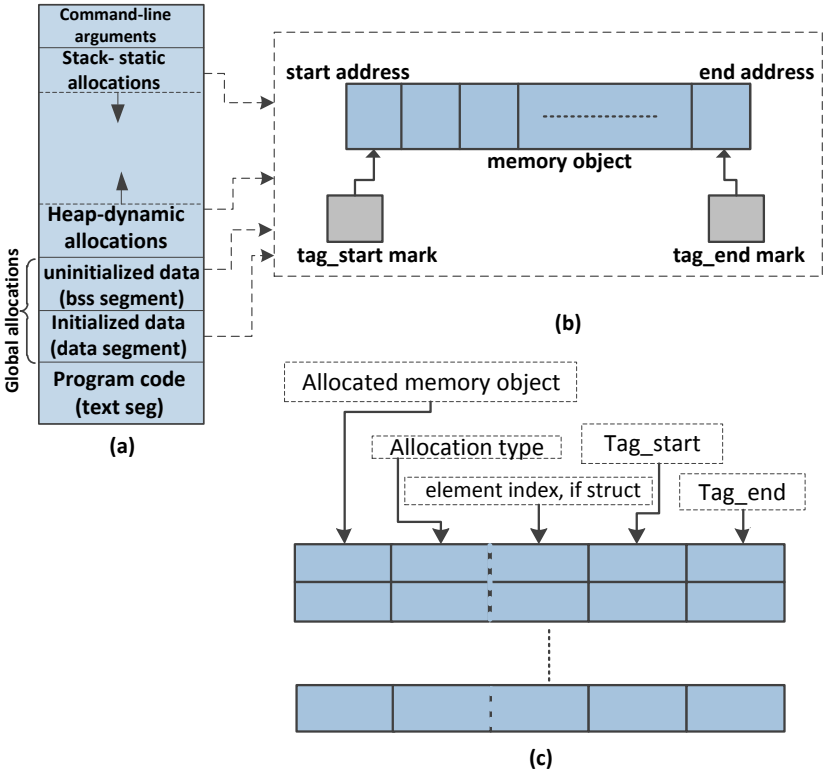

Fig. 8: (a) Typical memory layout of a C program. (b) Memory objects coupled with tag_start and tag_end marks. (c) Record table layout used at the time of code instrumentation.

Fig. 8(a). In order to instrument such global allocations the tag marks are created globally. For each global memory object that is allocated as an array data variable the tag_start and tag_end mark pointers are created. The start and end address of the given memory object are calculated and assign to the corresponding tag marks.

Step-3. Tag Creation for Local Memory Objects: For the local memory objects that are defined at function level statically, the memory is reserved explicitly on stack and tag marks are also allocated on stack for such memory objects. The start and end address of such memory objects are calculated, after which tag_start and tag_end marks are initialized respectively with these addresses. For each memory object pointer that is used to allocate memory dynamically, tag_start and tag_end mark pointers are also allocated and initialized with *NULL*. For such dynamically created objects, the memory is reserved on heap implicitly by calling special memory allocation functions (e.g, `malloc, calloc, realloc, xcalloc` etc.) and starting address is returned to a pointer variable. Our proposed solution intercepts such function calls and the start address is assigned to its tag_start mark pointer. End address of the allocated object is also determined by our solution and it is assigned to its corresponding tag_end mark pointer. If a pointer is derived from another pointer, the tag mark pointer associated with the actual object pointer must also be propagated. Our proposed technique detects store instructions, at LLVM-IR level, that are used to pass address values

ALGORITHM 1: Stage-5:Tag checks placement.

Input: Instrumented LLVM-IR code β_4 generated in stage-4; memory map table Tag_map_table;Dedicated tag address $globaltag$
Output: Final Instrumented LLVM-IR code γ generated through LLVM *opt* command using stage-5
for *each function definition* fun_def *in* β_3 **do**
  for *each instruction* fun_inst *in* fun_def **do**
    if fun_inst *is function call without definition and not a memory allocation or deallocation call* **then**
      for *each function argument* fun_arg *in* fun_inst **do**
        Create two memory objects $before\_fun$ and $after\_fun$. Retrieve respective tag_start and tag_end marks from Tag_map_table.
        Read address location next to tag_end address before and after fun_inst instruction and store the values in $before\_fun$ and $after\_fun$ respectively.
        Place tag check instruction after function call fun_inst comparing $before\_fun$ and $after\_fun$ memory objects.
      end
    end
    if fun_inst *is a STORE/LOAD instruction* **then**
      Retrieve respective tag_start and tag_end marks from Tag_map_table and get address to be accessed $address\_tobe\_accessed$ by the fun_inst instruction.
      Perform address comparison checks: $address\_tobe\_accessed$ with the tag_start and tag_end.
    end
  end
end
Delete memory map table Tag_map_table.
Save modified LLVM-IR code as an final instrumented LLVM-IR code γ

from one pointer object to another pointer object. Extra instructions are inserted by our solution to copy the tag mark of one pointer object to the tag mark of other pointer object. Our solutions instruments main memory objects and the sub-memory objects that are being allocated inside the *structs* type memory objects and it also instrument the sub-memory object that is being allocated inside another sub-memory object(such as linked lists).

Step-4. Inter-procedural Tag Propagation: As discussed earlier it is very critical to allocate tag marks in the same memory segment(e.g., heap, stack, bss, data) where memory object is being created. The memory objects can be accessed inside the body of another function through pointers that are passed as arguments at the function call. To handle inter-procedural tag marks propagation, functions containing pointers as arguments are duplicated. In order to update function calls for these newly created functions, the function call instructions are detected by our solution and replaced with new instructions so that tag marks can be propagated separately without changing the data-flow of the application under instrumentation.

Step-5. Tag Checks Placements: In the final step, the tag checks are created by following the steps as shown in Algorithm 1. Memory read and write accesses are performed through *LOAD* and *STORE* operations respectively at LLVM-IR level. Our solution detects such instructions and uses record table to locate the memory object pointer and tag marks to be accessed. Tag check instructions, to compare the start and end address of memory object with its associated tag marks, are then inserted before each load and store instruction to detect spatial IMA bug.

Table II: Effectiveness of the proposed tag-protection solution on different applications from BugBench benchmark suite

Application	Bug location	Bug type	Detected
bc-1.06	storage.c:177	heap	yes
bc-1.06	util.c:577	heap	yes
bc-1.06	bc.c:1425	global	yes
gzip-1.2.4	gzip.c:457	global	yes
man-1.5h1	man.c:978	global	yes
ncompress	compress.c:896	stack	yes
polymorph-0.40	polymorph.c:120	global	yes
polymorph-0.40	polymorph.c:193	stack	yes
squid-2.3	ftp.c:1024	heap	yes

3.2.1. Implementation. The proposed approach operates at the source-code level and it is loaded as an instrumentation *pass* at compile time. The current implementation is based on the LLVM v3.4 compiler infrastructure as shown in Fig. 9. The *Clang* compiler is used to compile C/C++ source file and generate Intermediate Representation (LLVM-IR) code. The LLVM Linker (llvm-link) is then used to link and generate a single LLVM-IR code file before running the instrumentation *pass*. In order to have minimum overhead, this *pass* is placed at the end of optimization pipeline and instruments only those memory operations that sustains other optimizations implemented by the LLVM Optimizer (opt). For instance, the memory operations such as accesses to local stack variables and objects created through LLVM code generator (e.g., debug information and metadata) will not be instrumented by our *pass* as these will be optimized out by the LLVM during pre-processing at compile time. After the LLVM-IR code has been transformed by our layer-2 instrumentation *pass* , it is processed again through LLVM optimization pipeline in order to simplify the tag marks propagation and checks. Furthermore, our solution is independent of any specific Instruction Set Architecture (ISA) as it is executed on LLVMs target-independent intermediate representation form.

We have also assessed our compile-time instrumentation solution using publicly available Wilander and Nikiforakis' benchmark software, runtime intrusion prevention evaluator (RIPE) [Wilander et al. 2011]. Various buffer-overflow vulnerabilities depending on the technique used to overflow the buffer, the kinds of attacks performed and the location of the buffer to be overwritten, have been covered by RIPE. For instance, RIPE covers four memory locations: Stack, Heap, BSS, and Data segment to allocate the buffer to be overflowed and uses return address, old base pointer, function pointer, *longjmp* buffers and buffers inside the *structs* to as code target pointers. Our solution provides 100% accuracy by successfully detecting all the supplied overflows.

To verify the effectiveness of second layer of IDPM, real-world applications, from BugBench benchmark suite [Lu et al. 2005] that have been reported with buffer overflow vulnerabilities, are also compiled and instrumented. These applications are then executed using input sets, triggering each known IMA bug. Our proposed solution detected all the bugs successfully as presented in Table II.

3.2.2. Limitations. Calls to pre-compiled library functions where source code is not available (e.g., `memcpy`, `strcpy`, `sscanf` etc.) are also identified. In such cases, it is not possible to insert tag address checking instructions. Alternatively, our code instrumentation solution inserts one tag value check instruction after such function calls by detecting memory objects passed as function arguments and loading tag mark values as defined in Algorithm 1. Any write overflow that occurs, as a result of sequential

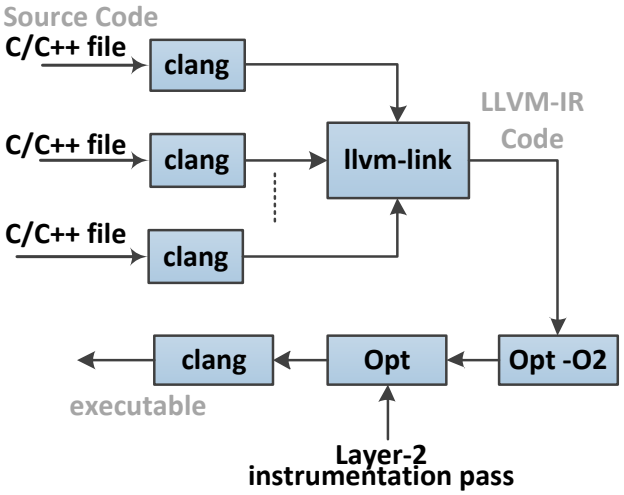

Fig. 9: Layer-2 implementation block diagram based on LLVM v3.4 compiler framework

write operation during an array or continuous memory access, during the execution of these functions will overwrite the memory pointed by the tag mark which will be eventually detected on function return by tag value check instructions placed after function call. If the overflow attack occurs at the same memory location with different values over a period of time (such as brute-force or adaptive attacks), the tag mark can be overwritten with the same value that was present initially. In such cases, our address checking mechanism can detect these kinds of attacks only if complete source code is provided.

4. EVALUATION AND EXPERIMENTAL RESULTS

The effectiveness and overheads are evaluated by implementing the proposed IDPM within the application running on the base station. In the first step, a malicious node is introduced into the system which compromises the base station and in the second step, it is shown that our solution can successfully detect and prevent such attacks with minimum impact on the system resources, performance and power consumption.

The first layer of IDPM is mainly responsible to detect performance degradation attacks such as detection of invalid packets transmitted by a malicious sensor node with the aim to drain battery and unnecessary utilization of system resources (i.e base station transceiver). Any data corruption as the result of buffer overflow will also be detected by this layer as long as the application's data memory is intact and program instructions are executing. However, it is observed that the attacker sensor node can initiate buffer overflow attack leading to data memory corruption where microcontroller failed to continue application execution. The second layer of the IDPM has successfully detected such buffer overflows when initiated by the attacker sensor node.

Through an experimental setup, as shown in Fig. 10, it is demonstrated that the given IoT system can be intruded if the attacker is familiar with the communication protocol used by the valid sensor nodes. As the sensor nodes are battery operated, the encryption support has been disabled to save the power. The effectiveness of the proposed solution has been tested under different attack scenarios as explained below.
Test Case 1: The first test case refers to inclusion of malicious sensor node with invalid

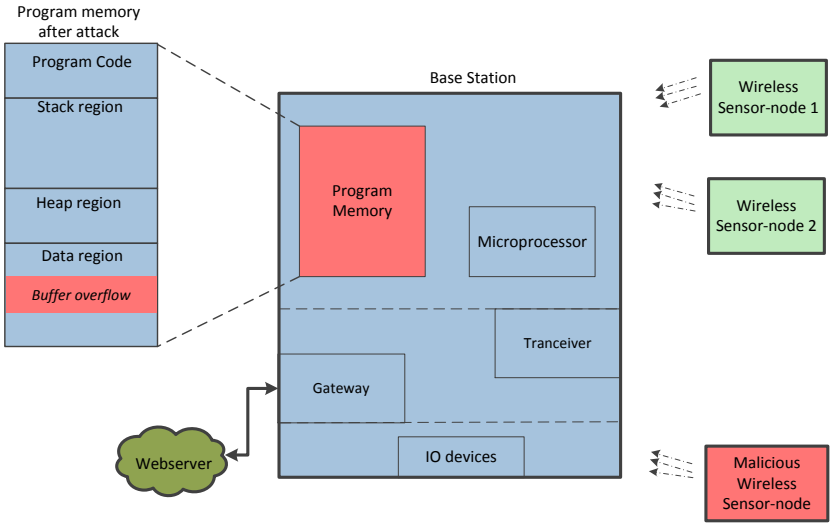

Fig. 10: Smart Controller IoT system compromised with malicious sensor node

node ids in order to drain the system battery and degrade the system performance. In such cases where the attacker sensor node is transmitting data with *node ids* different from authorized *node ids*. In such cases, the system output is not affected but the base station have to receive and discard packet each time consuming extra power.
Test Case 2: The attacker might be able to get valid *node ids* through eavesdropping by receiving the data packets being transmitted by the valid sensor nodes. In this case, the attacker sensor node can flood the base station by sending data packets with a valid node id, leading to performance degradation of the system. This will directly affect the aggregate execution time.
Test Case 3: The attacker can disable the valid sensor and replace it with another sensor node transmitting malicious packet with invalid *node id*. In this way, the attacker can send invalid packets to the base station causing degradation in the performance.
Test Case 4: In the worst case scenario, the attacker can disable the valid sensor and replace it with another sensor node transmitting malicious packet with valid *node id*. In this way, the attacker can send invalid values of *temperature,*CO_2 and *humidity* resulting incorrect working of the base station.
Test Case 5: If the length of the packet being transmitted by the valid sensor nodes is identified by the attacker, then buffer overflow based attacks can be launched by sending packets exceeding the valid data length and corrupting the program memory. In such cases, the stored data value can be overwritten which can alter actual flow of data leading to system malfunction.
Test Case 6: The valid sensor nodes might malfunction either due to depleting battery or faults in the sensor nodes. In such cases, the sensor node can transmit invalid data or completely fail to operate properly. Furthermore, if the base station transceiver stops receiving data, then system will not be able generate correct output. Our proposed solution also acts as sensor node health monitoring system. Such events will be reported to the main web server.

These test cases have been summarised in Table III. To practically evaluate these attack scenarios, we placed a malicious sensor node in the range of the base station. This malicious sensor node can transmit the packet containing more data as intended to receive or even transmitting packets containing invalid data. The length and format

Table III: Test cases under different attack scenarios

Test Cases	Attack Scenario
Test Case 1	Attacker can • Add extra sensor node • Have invalid *node id* • Transmit invalid sensor data
Test Case 2	Attacker can • Add extra sensor node • Have valid *node id* • Transmit invalid sensor data
Test Case 3	Attacker can • Disable one of the valid sensor nodes • Add malicious sensor node with invalid *node id*
Test Case 4	Attacker can • Disable one of the valid sensor nodes • Add malicious sensor node with valid *node id* • Transmit invalid sensor data
Test Case 5	Attacker can • Identify length of valid data packet • Generate buffer overflow attack at the base station
Test Case 6	• One of the valid sensor nodes stop transmitting data • Base station stops receiving data.

of data transmitted by valid sensor nodes is analysed by placing another receiving node. After that, the attacker sensor node starts transmitting its own packet leading to performance degradation and generating buffer overflow in the memory where the received data is being stored by the base station. In this way, the base station failed to execute the code in the correct manner. On the contrary, when the base station is protected with our proposed IDPM, the security attacks are detected successfully. In such event, the base station is configured to stop receiving further packets and an alarm signal is generated to the web server for further action.

The anomaly based intrusion detection systems (IDSs) work on the notion that any intrusive activity is a subset of anomalous behaviour of the system. If the attacker is not fully aware of the communication protocol followed by the valid sensor nodes and tries to intrude into the base station, then this will be detected as an anomalous activity by the system with high probability. In the worst case scenario, if the attacker knows the system very well, then it becomes more difficult to detect the intrusion. Kumar and Stafford [Kumar and Spafford 1994] suggested that the output of any anomaly based IDS can be categorized into four different groups(i.e. true positives, true negatives, false positives and false negatives). For instance, if there is an intrusion in the system and the IDS successfully detected it, then the IDS output will be labelled as a true positive (TP). False negatives (FN) represent those cases when the intrusion is left undetected. Similarly, if there is no intrusion in the system and IDS does not

Table IV: Accuracy of Proposed Intrusion Detection and Prevention Solution

Test Cases	True Positives	False Negatives	True Negatives	False Positives	Detection Level
Overall	96.52%	3.48%	97.94%	2.06%	Layer-1,2
Test Case 1	100%	0%			Layer-1
Test Case 2	98.41%	1.60%			Layer-1
Test Case 3	100%	0%			Layer-1
Test Case 4	94%	6%			Layer-1
Test Case 5	100%	0%			Layer-1,2
Test Case 6	100%	0%			Layer-1

generate any alarm signal, then it will be termed as a true negative (TN). Reporting normal behaviour of the system as intrusive is labelled as a false positive (FP).

The detection accuracy of the proposed IDPM is defined as its ability to detect normal behaviour and intrusion in the system successfully. This has been measured by using equation 12.

$$IDPM\ detection\ accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{12}$$

Table IV presents the detection accuracy of our proposed solution under different attack scenarios. Overall, our proposed IDPM has reported an accuracy of 97.23% with false negative rate of 3.48%. Majority of the false negatives are generated by *"Test Case 4"* where it is assumed that the attacker has complete knowledge and access to the sensor nodes. The valid sensor node has been replaced with a malicious sensor node having valid *node id*. On closer analysis of this test case results, it is the observed that if the invalid sensor node is transmitting data within the allowed frequency range and not all packet data values(i.e. temperature, CO_2, humidity) are invalid, then the IDPM will fail to detect it. Such cases are very hard to detect and normally labelled as non-intrusive as they very closely correlate to the normal behaviour of the system. Similarly, in the *"Test Case 2"*, the attack scenario corresponds to degradation of system performance by overloading the transceiver module and the IDPM has reported 1.60% false negatives. If the invalid node is not transmitting data frequently, then it will have minimum effect on the system execution time. In that case, it will be treated as normal behaviour by the IDPM. Beside these two test cases, our proposed security solution has presented 100% true positives. Overall, the IDPM has presented 2.06% false positives and correspond to those cases where valid sensor nodes transmit data values that are very close to maximum allowed range learned by the RNN model. *"Test Case 6"* presents those cases when the valid sensor node either develops a fault or it runs out of battery. The base station transceiver can also develop a fault and stop receiving data. In all such cases, our IDPM has presented 100% true positive results.

Beside verifying the effectiveness of the proposed IDPM, its impact on the system performance, power consumption, data transfer rate and hardware resources has been analysed. The performance overhead is measured in terms of system execution time that is defined as the time required to generate the output after receiving data from the transceiver module. Similarly, power consumption is defined as the power required to execute the single event and it has been calculated by measuring the current drawn by base station while running the application. Here, data transfer rate is defined the maximum number bits that can be received by the base station in one second. Hard-

Table V: Overhead of the proposed Intrusion Detection and Prevention Mechanism

Security Level	System Execution Time (ms)	Power Consumption (mW)	Data Transfer Rate ($kbps$)	Dedicated Hardware Logic	Packet Length$^{\beta}$ ($bytes$)
Baseline	8.584	72	31.69	–	255
AES Encrypted	8.623	86.90	31.54	YES	64
Change*	+0.45%	+20.69%	-0.54%		
Layer-1	8.794	78.545	30.93	NO	255
Change*	+2.45%	+9.09%	-2.40%		
Layer-2	8.644	72.821	31.47	NO	255
Change*	+0.7%	+1.14%	-0.69%		
Combined$^{\alpha}$	8.868	79.524	30.67	NO	255
Change*	+3.31%	+10.45%	-3.22%		

*Percentage change with respect to baseline, $^{\beta}$Maximum length of a packet that can be received/-transmitted in a single event, $^{\alpha}$Proposed IDPM combining first and second layer.

ware resources overhead is defined as the requirement of dedicated hardware logic to implement the proposed solution. In the first step, these values have been measured for the baseline application running on the base station without encryption and IDPM. In the next step, these values are measured with encryption enabled, layer-1 only, layer-2 only and finally for the IDPM combining two layers respectively. From the results, as shown in Table V, it is visible that, the proposed IDPM has presented very minimal impact on the system.

"RFM 69 W" transceiver module on the base station has 128-bit AES encryption support. This has been implemented using dedicated hardware logic whereas our IDPM does not require any such dedicated resources. Although, the encryption enabled base station has lower execution time but it has higher power consumption as compared to our proposed IDPM. The encrypted communication also have a significant impact on the sensor nodes power consumption as each node has to transmit encrypted data, consuming more power for each sensor node in the system. Moreover, this encryption is only supported to transmit/receive packets with maximum data length of 64 bytes [HOPERF 2015]. Therefore, to transfer large packets, the power consumption of the encryption enabled base station will increase. On the contrary, our proposed IDPM has no such restrictions as it does not depend on the packet data length.

5. CONCLUSION

In this work, we have presented a multi-layer and effective intrusion detection and prevention mechanism for low-power IoT systems. In the first layer, an intelligent anomaly detection model is learned using RNN to deal with performance degradation attacks. In the second layer, a lightweight compile-time code instrumentation technique is implemented to protect program memory from illegal memory accesses. The proposed solution also acts as sensor node health monitoring system and it is shown that it can successfully detect the failure of a valid sensor node. The feasibility of the proposed solution is demonstrated for a wireless sensor nodes based IoT system, with the detection accuracy of 97.23%. The effectiveness of the proposed solution is further tested by adding attacker sensor node and generating different security attacks that are eventually detected by our solution. The proposed IDPM does not require dedi-

cated hardware resources and presented negligible performance overhead with 10.45% increase in the power consumption.

REFERENCES

Hossam Abdelbaki, Erol Gelenbe, and Said E El-Khamy. 2000. Analog hardware implementation of the random neural network model. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, Vol. 4. IEEE, 197–201.

J Aguilar and A Colmenares. 1998. Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm. *Pattern Analysis and Applications* 1, 1 (1998), 52–61.

Periklis Akritidis, Manuel Costa, Miguel Castro, and Steven Hand. 2009. Baggy Bounds Checking: An Efficient and Backwards-Compatible Defense against Out-of-Bounds Errors.. In *USENIX Security Symp.* 51–66.

Vicente Alarcon-Aquino, Javier Barria, and others. 2006. Multiresolution FIR neural-network-based learning algorithm applied to network traffic prediction. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 36, 2 (2006), 208–220.

Syed Obaid Amin, Muhammad Shoaib Siddiqui, Choong Seon Hong, and Sungwon Lee. 2009. RIDES: Robust intrusion detection system for IP-based ubiquitous sensor networks. *Sensors* 9, 5 (2009), 3447–3468.

Kumar Avijit and Prateek Gupta. 2006. Binary rewriting and call interception for efficient runtime protection against buffer overflows. *Software: Practice and Experience* 36, 9 (2006), 971–998.

Kumar Avijit, Prateek Gupta, and Deepak Gupta. 2004. TIED, LibsafePlus: Tools for Runtime Buffer Overflow Protection.. In *USENIX Security Symposium*. 45–56.

M.H. Bhuyan, D.K. Bhattacharyya, and J.K. Kalita. 2014. Network Anomaly Detection: Methods, Systems and Tools. *Communications Surveys Tutorials, IEEE* 16, 1 (First 2014), 303–336.

I. Butun, S.D. Morgera, and R. Sankar. 2014. A Survey of Intrusion Detection Systems in Wireless Sensor Networks. *Communications Surveys Tutorials, IEEE* 16, 1 (First 2014), 266–282.

C. Callegari, S. Giordano, and M. Pagano. 2014. Neural network based anomaly detection. In *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on*. 310–314. DOI:http://dx.doi.org/10.1109/CAMAD.2014.7033256

Dinakar Dhurjati, Sumant Kowshik, and Vikram Adve. 2006. SAFECode: Enforcing Alias Analysis for Weakly Typed Languages. In *Proceedings of the 27th ACM SIGPLAN Conference on Programm. Lang. Design and Imp.* ACM, New York, NY, USA, 144–157. DOI:http://dx.doi.org/10.1145/1133981.1133999

Ioannis Doudalis, James Clause, Guru Venkataramani, Milos Prvulovic, and Alessandro Orso. 2012. Effective and Efficient Memory Protection Using Dynamic Tainting. *Computers, IEEE Trans. on* 61, 1 (2012), 87–100.

Erol Gelenbe. 1989. Random neural networks with negative and positive signals and product form solution. *Neural computation* 1, 4 (1989), 502–510.

Erol Gelenbe. 1990. Stability of the random neural network model. *Neural computation* 2, 2 (1990), 239–247.

Erol Gelenbe. 1991. Product-form queueing networks with negative and positive customers. *Journal of applied probability* (1991), 656–663.

Erol Gelenbe. 1993. Learning in the recurrent random neural network. *Neural Computation* 5, 1 (1993), 154–164.

E. Gelenbe and K.F. Hussain. 2002. Learning in the multiple class random neural network. *Neural Networks, IEEE Transactions on* 13, 6 (Nov 2002), 1257–1267.

Michael Georgiopoulos, Cong Li, and Taskin Kocak. 2011. Learning in the feed-forward random neural network: A critical review. *Performance Evaluation* 68, 4 (2011), 361–384.

J. Granjal, E. Monteiro, and J. Sa Silva. 2015. Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues. *Communications Surveys Tutorials, IEEE* 17, 3 (thirdquarter 2015), 1294–1312. DOI:http://dx.doi.org/10.1109/COMST.2015.2388550

Sang-Jun Han and Sung-Bae Cho. 2005. Evolutionary neural networks for anomaly detection based on the behavior of a program. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 36, 3 (2005), 559–570.

Niranjan Hasabnis, Ashish Misra, and R. Sekar. 2012. Light-weight Bounds Checking. In *Proceedings of the Tenth International Symposium on CGO (CGO '12)*. ACM, New York, NY, USA, 135–144. DOI:http://dx.doi.org/10.1145/2259016.2259034

HOPERF. Accessed: 15-9-2015. RFM69 ISM TRANSCEIVER MODULE. (Accessed: 15-9-2015). http://www.hoperf.cn/upload/rf/RFM69-V1.3.pdf

A. Javed, H. Larijani, A. Ahmadinia, R. Emmanuel, D. Gibson, and C. Clark. 2015. Experimental testing of a random neural network smart controller using a single zone test chamber. *Networks, IET* 4, 6 (2015), 350–358. DOI:http://dx.doi.org/10.1049/iet-net.2015.0020

A. Javed, H. Larijani, A. Ahmadinia, and D. Gibson. 2016. Smart Random Neural Network Controller for HVAC using Cloud Computing Technology. *IEEE Transactions on Industrial Informatics* PP, 99 (2016), 1–1. DOI:http://dx.doi.org/10.1109/TII.2016.2597746

Richard WM Jones and Paul HJ Kelly. 1997. Backwards-Compatible Bounds Checking for Arrays and Pointers in C Programs.. In *Proceedings of the 3rd Int. Workshop on Automatic Debugging*. Citeseer, 13–26.

Georgios Kornaros and Dionisios Pnevmatikatos. 2013. A survey and taxonomy of on-chip monitoring of multicore systems-on-chip. *ACM Trans. Des. Autom. Electron. Syst.* 18, 2, Article 17 (2013), 38 pages.

Sandeep Kumar and Eugene H. Spafford. 1994. *An Application of Pattern Matching in Intrusion Detection*. Technical Report. Department of Computer Sciences, Purdue University.

Wenchao Li, Ping Yi, Yue Wu, Li Pan, and Jianhua Li. 2014. A new intrusion detection system based on KNN classification algorithm in wireless sensor network. *Journal of Elect. and Comp. Engineering* (2014).

Aristidis Likas and Andreas Stafylopatis. 2000. Training the random neural network using quasi-Newton methods. *European Journal of Operational Research* 126, 2 (2000), 331–339.

Shan Lu, Zhenmin Li, Feng Qin, Lin Tan, Pin Zhou, and Yuanyuan Zhou. 2005. Bugbench: Benchmarks for evaluating bug detection tools. In *Workshop on the Evaluation of Software Defect Detection Tools*. 1–5.

Shufu Mao and T. Wolf. 2010. Hardware Support for Secure Processing in Embedded Systems. *Computers, IEEE Transactions on* 59, 6 (2010), 847–854.

Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. 2012. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks* 10, 7 (2012), 1497 – 1516. DOI:http://dx.doi.org/10.1016/j.adhoc.2012.02.016

Samir Mohamed and Gerardo Rubino. 2002. A study of real-time packet video quality using random neural networks. *Circuits and Systems for Video Technology, IEEE Transactions on* 12, 12 (2002), 1071–1083.

Santosh Nagarakatte, Jianzhou Zhao, Milo MK Martin, and Steve Zdancewic. 2009. SoftBound: highly compatible and complete spatial memory safety for c. In *ACM Sigplan Notices*, Vol. 44. ACM, 245–258.

George C Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. 2005. CCured: type-safe retrofitting of legacy software. *ACM Trans. Program. Lang. Syst.* 27, 3 (2005), 477–526.

NIST. 2001. Advanced Encryption Standard:U.S. National Institute of Standards and Technology (NIST): Federal Information Processing Standards Publication (FIPS PUBS) 197. (2001). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Aleph One. 1996. Smashing the stack for fun and profit. *Phrack magazine* 7, 49 (1996), 14–16.

M. Rahmatian, H. Kooti, I.G. Harris, and E. Bozorgzadeh. 2012. Hardware-Assisted Detection of Malicious Software in Embedded Systems. *Embedded Systems Letters, IEEE* 4, 4 (2012), 94–97.

Shahid Raza, Linus Wallgren, and Thiemo Voigt. 2013. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad Hoc Networks* 11, 8 (2013), 2661–2674.

RSA. 2003. Public-Key Cryptography Standards (PKCS): RSA Cryptography Specifications Version 2.1. (2003). https://tools.ietf.org/html/rfc3447

Olatunji Ruwase and Monica S Lam. 2004. A Practical Dynamic Buffer Overflow Detector. In *In Proceedings of the 11th Annual Network and Distributed System Security Symposium*.

Edward J Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 317–331.

Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. 2012. AddressSanitizer: A fast address sanity checker. In *USENIX ATC*, Vol. 2012.

G. Edward Suh, Jae W. Lee, David Zhang, and Srinivas Devadas. 2004. Secure Program Execution via Dynamic Information Flow Tracking. *SIGARCH Comput. Archit. News* 32, 5 (Oct. 2004), 85–96.

Stelios Timotheou. 2008. Nonnegative least squares learning for the random neural network. In *Artificial Neural Networks-ICANN 2008*. Springer, 195–204.

Stelios Timotheou. 2010. The random neural network: a survey. *The computer journal* 53, 3 (2010), 251–267.

W. Trappe, R. Howard, and R.S. Moore. 2015. Low-Energy Security: Limits and Opportunities in the Internet of Things. *Security Privacy, IEEE* 13, 1 (Jan 2015), 14–21. DOI:http://dx.doi.org/10.1109/MSP.2015.7

Kleber Vieira, Alexandre Schulter, Carlos Westphall, and Carla Westphall. 2010. Intrusion Detection for Grid and Cloud Computing. *IT Professional* 12, 4 (2010), 38–43. DOI:http://dx.doi.org/10.1109/MITP.2009.89

John Wilander, Nick Nikiforakis, Yves Younan, Mariam Kamkar, and Wouter Joosen. 2011. RIPE: Runtime Intrusion Prevention Evaluator. In *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM.

Shelly Xiaonan Wu and Wolfgang Banzhaf. 2010. The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing* 10, 1 (2010), 1 – 35. DOI:http://dx.doi.org/10.1016/j.asoc.2009.06.019

Li Da Xu, Wu He, and Shancang Li. 2014. Internet of Things in Industries: A Survey. *Industrial Informatics, IEEE Transactions on* 10, 4 (Nov 2014), 2233–2243. DOI:http://dx.doi.org/10.1109/TII.2014.2300753

Man-Ki Yoon, S. Mohan, Jaesik Choi, Jung-Eun Kim, and Lui Sha. 2013. SecureCore: A multicore-based intrusion detection architecture for real-time embedded systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*. 21–32.

Yves Younan. 2014. 25 Years of Vulnerabilities: 1988-2012. (2014). http://labs.snort.org/blogfiles/Sourcefire\ \-25-Years-of-Vulnerabilities-Research-Report.pdf.

Yves Younan, Pieter Philippaerts, Lorenzo Cavallaro, R Sekar, Frank Piessens, and Wouter Joosen. 2010. PAriCheck: an efficient pointer arithmetic checker for C programs. In *Proceedings of the 5th ACM Symp. on Info., Comp. and Comm. Security*. ACM, 145–156.